datalog

Datomic

agenda

Datomic and datalog

joins

functional extensions

navigational extensions

disjunction and negation

chronological extensions

programming with data

what is Datomic?

ACID database (consistent)

datalog query (powerful, ad hoc, joins)

scalable read (distribute load)

universal schema (flexible, hierarchical, sparse)

indelible and time-aware (remembers stuff)

what is Datomic?

ACID database (consistent)

datalog query (powerful, ad hoc, joins)

scalable read (distribute load)

universal schema (flexible, hierarchical, sparse)

indelible and time-aware (remembers stuff)

what is datalog?

deductive query system

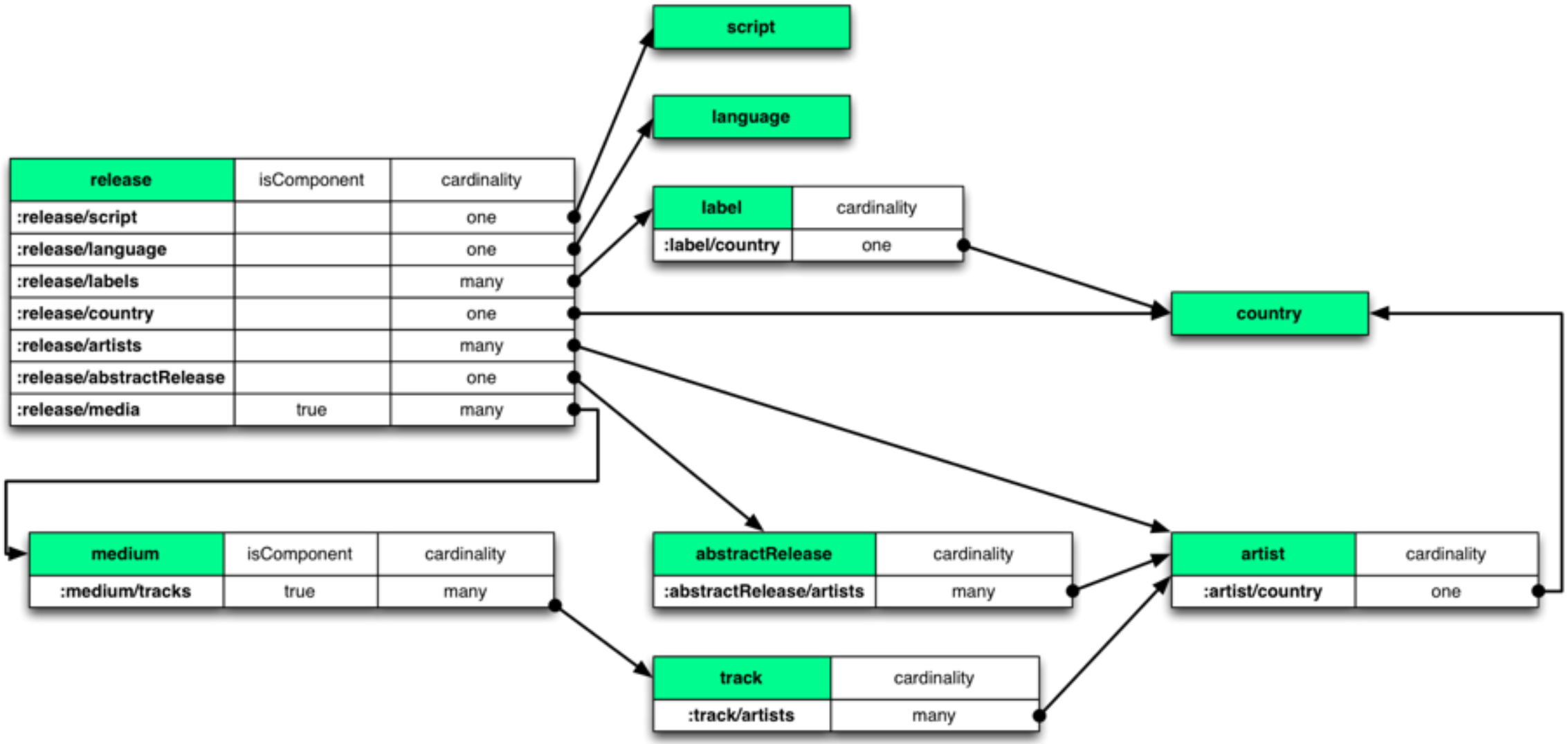
database of facts

rules for deriving new information

finds facts from partial specifications

works extremely well w/ OO and sexpr languages

example: mbrainz



https://github.com/Datomic/mbrainz-sample

a little Datomic datalog

facts are five-tuples

entity, attribute, value, transaction, op

basic specification via *data clauses*

constants constrain *results*, variables bound to matching tuples

find arguments get result data out

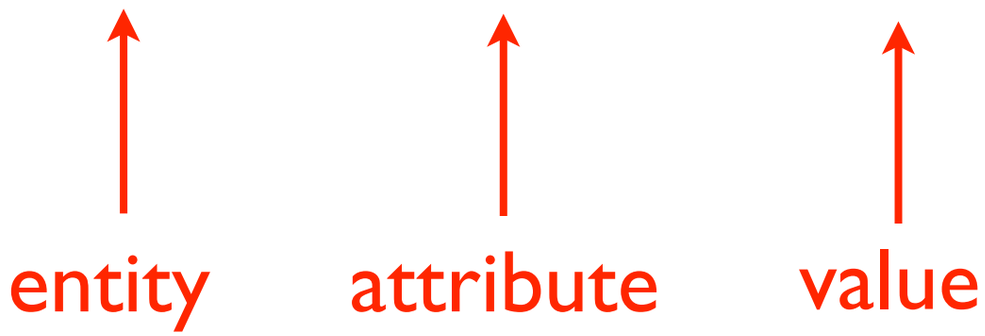
in arguments put input data in

database subset

entity	attribute	value
42	:release/name	Led Zeppelin
43	:release/name	Led Zeppelin II
42	:release/artists	107
43	:release/artists	107

OO syntax

```
release1.name("Led Zeppelin");
release2.name("Led Zeppelin II");
release1.artists(artist);
release2.artists(artist);
```



matching a data clause

entity	attribute	value
42	:release/name	Led Zeppelin
43	:release/name	Led Zeppelin II
42	:release/artists	107
43	:release/artists	107

```
release.name(name);
```

a complete query

returns

```
Query q = find(name)
          .where(release.name(name));
```

specifications

vars

leverage type systems

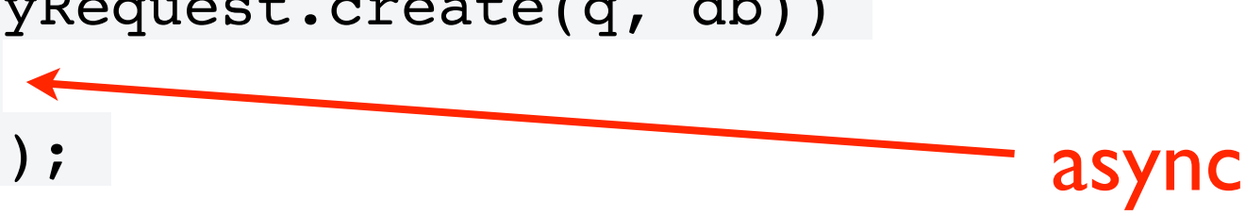
```
static final Release.V release = Release.var();
static final Var<String> name = var();
```

```
Query q = find(name)
          .where(release.name(name));
```

consuming query results

```
Database db = conn.db();
Query q = find(name)
          .where(release.name(name));
Collection<Tuple> value =
client.q(QueryRequest.create(q, db))
        .get()
        .value();

for (Iterator<Tuple> it = value.iterator(); it.hasNext(); ) {
    Tuple t = it.next();
    System.out.println(t.get(name));
}
```



consuming query results

```
Database db = conn.db();
Query q = find(name)
          .where(release.name(name));
Collection<Tuple> value =
client.q(QueryRequest.create(q, db))
        .get()
        .value();

for (Iterator<Tuple> it = value.iterator(); it.hasNext(); ) {
    Tuple t = it.next();
    System.out.println(t.get(name));
}
```

plain ol' collection

consuming query results

```
Database db = conn.db();
Query q = find(name)
          .where(release.name(name));
Collection<Tuple> value =
client.q(QueryRequest.create(q, db))
        .get()
        .value();

for (Iterator<Tuple> it = value.iterator(); it.hasNext(); ) {
    Tuple t = it.next();
    System.out.println(t.get(name));
}
```

vars link items in spec & in result tuple

constant V

entity	attribute	value
42	:release/name	Led Zeppelin
43	:release/name	Led Zeppelin II
42	:release/artists	107
43	:release/artists	107

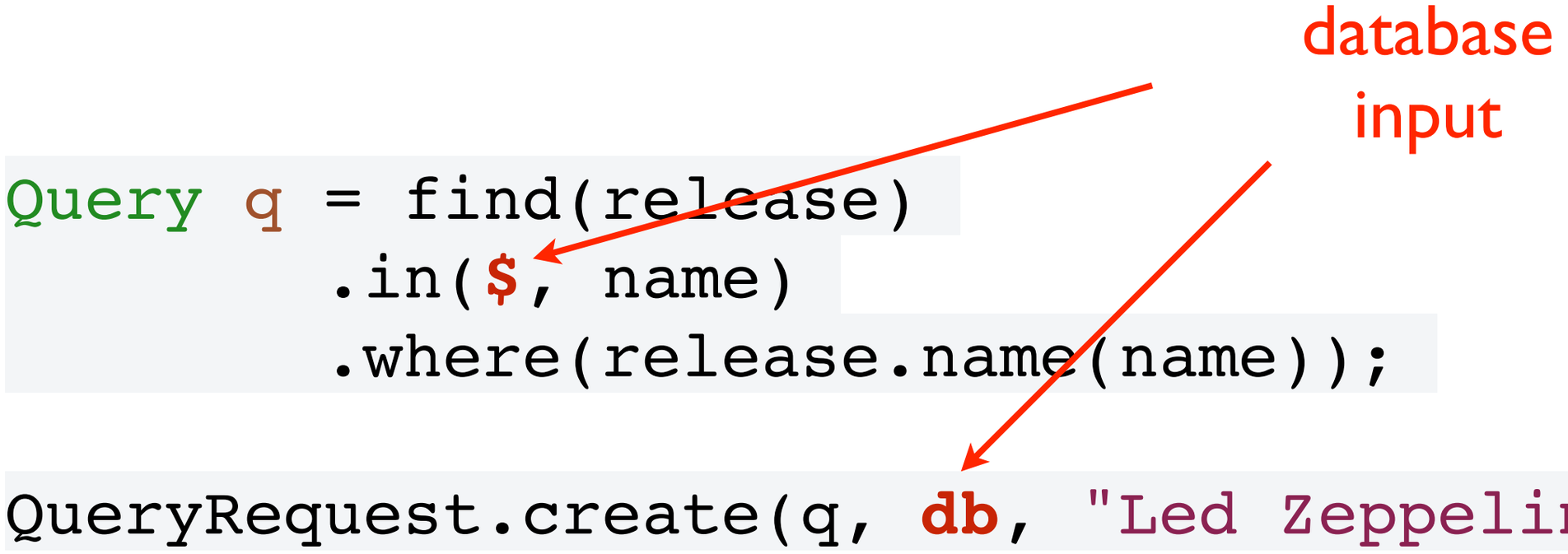
```
find(release)
.where(release.name("Led Zeppelin"));
```

in args

```
Query q = find(release)
          .in($, name)
          .where(release.name(name));

QueryRequest.create(q, db, "Led Zeppelin"))
```

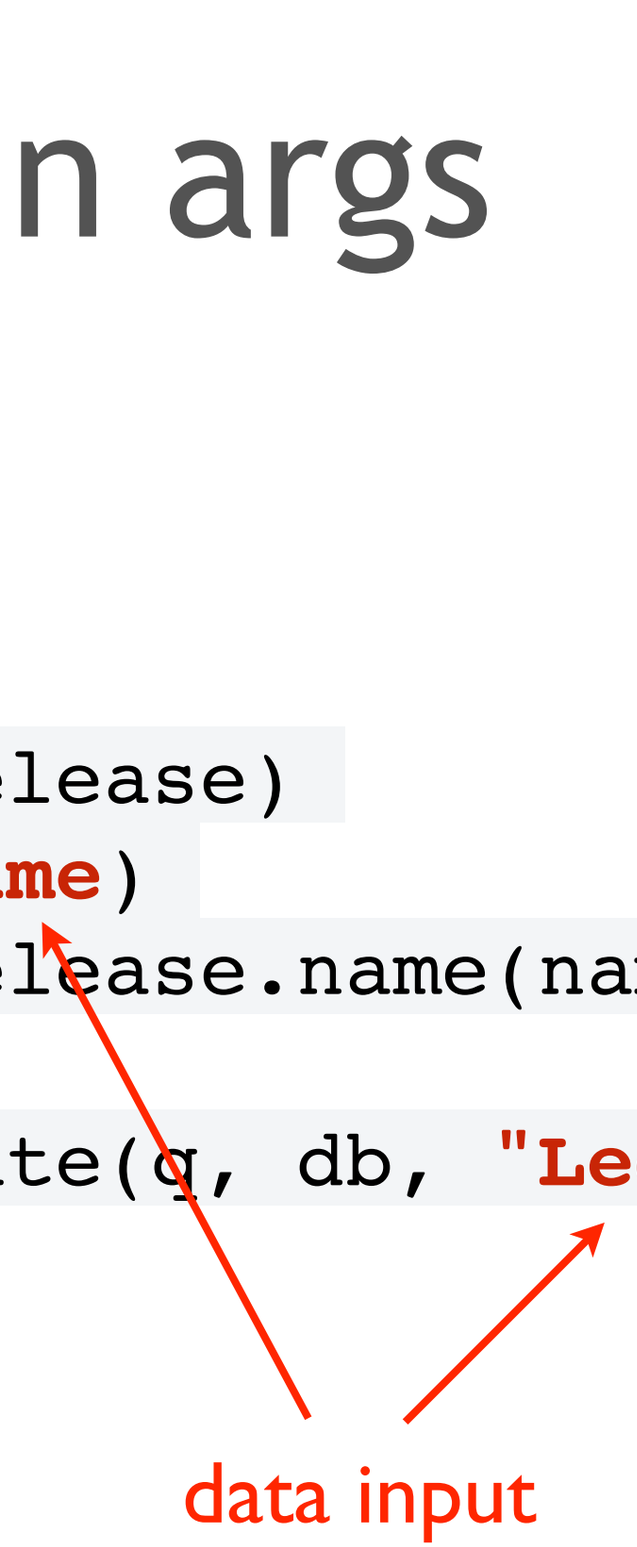
in args


database
input
Query q = find(release)
.in($, name)
.where(release.name(name));
QueryRequest.create(q, db, "Led Zeppelin"))

in args

```
Query q = find(release)
          .in($, name)
          .where(release.name(name));

QueryRequest.create(q, db, "Led Zeppelin"))
```



joins

implicit (repeated vars)

universal schema **FTW**

- flat
  - no structural navigation
  - no nesting / containment
  - no directionality penalties

a simple join

```
Query q = find(releaseName)
     .where(release.name(releaseName),
            release.artists(artist),
            artist.name(artistName));
```

just repeat variables!

```
Query q = find(releaseName)
     .where(release.name(releaseName),
            release.artists(artist),
            artist.name(artistName));
```

no table structure

```
release.artists(artist)
```

datalog

```
.join(ARTIST)
.on(ARTIST.ID.equals(RELEASE.ARTIST_ID)
```

SQL

a four-way join

```
find(title, album, year)
.in($, artistName)
.where(artist.name(artistName),
       track.artists(artist),
       track.name(title),
       medium.tracks(track),
       release.media(medium),
       release.name(album),
       release.year(year));
```

avoid documents

powerless

inflexible

choice != flexibility

documents force you into choices for relationships

embed relations?

links?

which direction?

choices must be made again for every relation

queries must reference these choices explicitly

need a different query? start over and model again!

functional extensions

predicates

aggregates

comparison predicate

```
find(artistCount)
.in($, limitYear)
.where(artist.startYear(startYear),
       Op.lt(startYear, limitYear),
       artist.name());
```

aggregate expression

aggregate in find

```
Var<Long> releaseCount = var();
Query q = find(Op.count(release).as(releaseCount))
    .where(artist.name("Paul McCartney"),
           release.artists(artist),
          release.name(name));

tuple.get(releaseCount);
```

find via “as” var

pull expressions

pull provides declarative navigation

datalog finds entities, then …

… pull navigates from entities to/through attributes

pull specific attributes (with types)

pull wildcards

pull recursively

pull example

```
Query q = find(artist._pull(Artist.startYear,
                            Artist.endYear),
               release._pull(ALL))
          .where(artist.name("Paul McCartney"),
                 release.artists(artist));
```

```
for (Iterator<Tuple> iterator = tuples.iterator();
     iterator.hasNext(); ) {
    Tuple t = iterator.next();
    Artist.E a = t.get(artist);
    Release.E r = t.get(release);
    System.out.println(a.startYear() + "  " +
                       a.endYear() + "  " +
                       r.name() + "  " +
                       r.media());
}
```

pull example

```
Query q = find(artist._pull(Artist.startYear,
                            Artist.endYear),
               release._pull(ALL))
          .where(artist.name("Paul McCartney"),
                 release.artists(artist));
```

explicit attrs

```
for (Iterator<Tuple> iterator = tuples.iterator();
     iterator.hasNext(); ) {
    Tuple t = iterator.next();
    Artist.E a = t.get(artist);
    Release.E r = t.get(release);
    System.out.println(a.startYear() + "  " +
                       a.endYear() + "  " +
                       r.name() + "  " +
                       r.media());
}
```

pull example

```
Query q = find(artist._pull(Artist.startYear,
                            Artist.endYear),
               release._pull(ALL))
          .where(artist.name("Paul McCartney"),
                 release.artists(artist));
```

all attrs

```
for (Iterator<Tuple> iterator = tuples.iterator();
     iterator.hasNext(); ) {
    Tuple t = iterator.next();
    Artist.E a = t.get(artist);
    Release.E r = t.get(release);
    System.out.println(a.startYear() + "  " +
                       a.endYear() + "  " +
                       r.name() + "  " +
                       r.media());
}
```

pull example

```
Query q = find(artist._pull(Artist.startYear,
                            Artist.endYear),
               release._pull(ALL))
          .where(artist.name("Paul McCartney"),
                 release.artists(artist));
```

```
for (Iterator<Tuple> iterator = tuples.iterator();
     iterator.hasNext(); ) {
    Tuple t = iterator.next();
    Artist.E a = t.get(artist);
    Release.E r = t.get(release);
    System.out.println(a.startYear() + "  " +
                       a.endYear() + "  " +
                       r.name() + "  " +
                       r.media());
}
```

entity types

pull example

```
Query q = find(artist._pull(Artist.startYear,
                            Artist.endYear),
               release._pull(ALL))
          .where(artist.name("Paul McCartney"),
                 release.artists(artist));
```

```
for (Iterator<Tuple> iterator = tuples.iterator();
     iterator.hasNext(); ) {
    Tuple t = iterator.next();
    Artist.E a = t.get(artist);
    Release.E r = t.get(release);
    System.out.println(a.startYear() + "  " +
                       a.endYear() + "  " +
                       r.name() + "  " +
                       r.media());
}
```

disjunction

```
Query q = find(Op.count(artist).as(count))
     .where(artist.name(),
          or(artist.country(Country.US),
             artist.country(Country.CA)));
```

nest clauses
inside **or**

negation

match some tuples

```
q = find(Op.count(artist).as(count))
    .where(artist.name(),
           not(artist.country(Country.US)));
```

shrink match-in-hand by subtracting a different match

agile queries

style	supporting index
k/v	AVET
row (RDBMS)	EAVT
column	AEVT
document	EAVT, components
graph	VAET

indelible and chronological

E / A / V / **Transaction / Added**

transactions are first class entities

db is accumulate-only

“deletion” accumulates a datom with Added: false

not append-only

apply time travel filters to db without changing query

chronological filters

db view	semantics	supports
(default)	current state	what is the current situation?
.asOf	state at point in past	how were things in the past?
.since	state since point in past	how have things changed?
tx report	before / after / change view of tx	automated event response
.with	state with proposed additions	what would happen if we did X?
.history	timeless view of all history	anything!

asOf query

```
Database db = conn.db();
Database olderDb =
db.asOf(formatter.parse("01/01/2013"));
```

filter to a different point in time

```
Query q = find(…);
Collection<Tuple> tuples =
    client.q(QueryRequest.create(q, olderDb))
    .get()
    .value();
```

only input changes, query remains same

raw data

e	a	v	tx	added
0x00000c00000003e9	:db/txInstant	Mon Dec 31 19:00:00	0xc00000003e9	TRUE
0x00001000000003ea	:item/id	DLC-042	0xc00000003e9	TRUE
0x00001000000003ea	:item/description	Dilitihium Crystals	0xc00000003e9	TRUE
0x00001000000003ea	:item/count	100	0xc00000003e9	TRUE
0x00000c00000003eb	:db/txInstant	Thu Jan 31 19:00:00	0xc00000003eb	TRUE
0x00001000000003ea	:item/count	100	0xc00000003eb	FALSE
0x00001000000003ea	:item/count	250	0xc00000003eb	TRUE
0x00000c00000003ec	:db/txInstant	Thu Feb 27 19:00:00	0xc00000003ec	TRUE
0x00001000000003ea	:item/count	250	0xc00000003ec	FALSE
0x00001000000003ea	:item/count	50	0xc00000003ec	TRUE
0x00000c00000003ed	:db/txInstant	Mon Mar 31 20:00:00	0xc00000003ed	TRUE
0x00000c00000003ed	:tx/error	TRUE	0xc00000003ed	TRUE
0x00001000000003ea	:item/count	50	0xc00000003ed	FALSE
0x00001000000003ea	:item/count	9999	0xc00000003ed	TRUE
0x00000c00000003ee	:db/txInstant	Wed May 14 20:00:00	0xc00000003ee	TRUE
0x00001000000003ea	:item/count	9999	0xc00000003ee	FALSE
0x00001000000003ea	:item/count	100	0xc00000003ee	TRUE

default filter

e	a	v	tx	added
0x00000c00000003e9	:db/txInstant	Mon Dec 31 19:00:00	0xc00000003e9	TRUE
0x00001000000003ea	:item/id	DLC-042	0xc00000003e9	TRUE
0x00001000000003ea	:item/description	Dilitihium Crystals	0xc00000003e9	TRUE
0x00001000000003ea	:item/count	100	0xc00000003e9	TRUE
0x00000c00000003eb	:db/txInstant	Thu Jan 31 19:00:00	0xc00000003eb	TRUE
0x00001000000003ea	:item/count	100	0xc00000003eb	FALSE
0x00001000000003ea	:item/count	250	0xc00000003eb	TRUE
0x00000c00000003ec	:db/txInstant	Thu Feb 27 19:00:00	0xc00000003ec	TRUE
0x00001000000003ea	:item/count	250	0xc00000003ec	FALSE
0x00001000000003ea	:item/count	50	0xc00000003ec	TRUE
0x00000c00000003ed	:db/txInstant	Mon Mar 31 20:00:00	0xc00000003ed	TRUE
0x00000c00000003ed	:tx/error	TRUE	0xc00000003ed	TRUE
0x00001000000003ea	:item/count	50	0xc00000003ed	FALSE
0x00001000000003ea	:item/count	9999	0xc00000003ed	TRUE
0x00000c00000003ee	:db/txInstant	Wed May 14 20:00:00	0xc00000003ee	TRUE
0x00001000000003ea	:item/count	9999	0xc00000003ee	FALSE
0x00001000000003ea	:item/count	100	0xc00000003ee	TRUE

as-of filter

e	a	v	tx	added
0x00000c00000003e9	:db/txInstant	Mon Dec 31 19:00:00	0xc00000003e9	TRUE
0x00001000000003ea	:item/id	DLC-042	0xc00000003e9	TRUE
0x00001000000003ea	:item/description	Dilitihium Crystals	0xc00000003e9	TRUE
0x00001000000003ea	:item/count	100	0xc00000003e9	TRUE
0x00000c00000003eb	:db/txInstant	Thu Jan 31 19:00:00	0xc00000003eb	TRUE
0x00001000000003ea	:item/count	100	0xc00000003eb	FALSE
0x00001000000003ea	:item/count	250	0xc00000003eb	TRUE
0x00000c00000003ec	:db/txInstant	Thu Feb 27 19:00:00	0xc00000003ec	TRUE
0x00001000000003ea	:item/count	250	0xc00000003ec	FALSE
0x00001000000003ea	:item/count	50	0xc00000003ec	TRUE
0x00000c00000003ed	:db/txInstant	Mon Mar 31 20:00:00	0xc00000003ed	TRUE
0x00000c00000003ed	:tx/error	TRUE	0xc00000003ed	TRUE
0x00001000000003ea	:item/count	50	0xc00000003ed	FALSE
0x00001000000003ea	:item/count	9999	0xc00000003ed	TRUE
0x00000c00000003ee	:db/txInstant	Wed May 14 20:00:00	0xc00000003ee	TRUE
0x00001000000003ea	:item/count	9999	0xc00000003ee	FALSE
0x00001000000003ea	:item/count	100	0xc00000003ee	TRUE

since filter

e	a	v	tx	added
0x00000c00000003e9	:db/txInstant	Mon Dec 31 19:00:00	0xc00000003e9	TRUE
0x00001000000003ea	:item/id	DLC-042	0xc00000003e9	TRUE
0x00001000000003ea	:item/description	Dilitihium Crystals	0xc00000003e9	TRUE
0x00001000000003ea	:item/count	100	0xc00000003e9	TRUE
0x00000c00000003eb	:db/txInstant	Thu Jan 31 19:00:00	0xc00000003eb	TRUE
0x00001000000003ea	:item/count	100	0xc00000003eb	FALSE
0x00001000000003ea	:item/count	250	0xc00000003eb	TRUE
0x00000c00000003ec	:db/txInstant	Thu Feb 27 19:00:00	0xc00000003ec	TRUE
0x00001000000003ea	:item/count	250	0xc00000003ec	FALSE
0x00001000000003ea	:item/count	50	0xc00000003ec	TRUE
0x00000c00000003ed	:db/txInstant	Mon Mar 31 20:00:00	0xc00000003ed	TRUE
0x00000c00000003ed	:tx/error	TRUE	0xc00000003ed	TRUE
0x00001000000003ea	:item/count	50	0xc00000003ed	FALSE
0x00001000000003ea	:item/count	9999	0xc00000003ed	TRUE
0x00000c00000003ee	:db/txInstant	Wed May 14 20:00:00	0xc00000003ee	TRUE
0x00001000000003ea	:item/count	9999	0xc00000003ee	FALSE
0x00001000000003ea	:item/count	100	0xc00000003ee	TRUE

history = raw

e	a	v	tx	added
0x00000c00000003e9	:db/txInstant	Mon Dec 31 19:00:00	0xc00000003e9	TRUE
0x00001000000003ea	:item/id	DLC-042	0xc00000003e9	TRUE
0x00001000000003ea	:item/description	Dilitihium Crystals	0xc00000003e9	TRUE
0x00001000000003ea	:item/count	100	0xc00000003e9	TRUE
0x00000c00000003eb	:db/txInstant	Thu Jan 31 19:00:00	0xc00000003eb	TRUE
0x00001000000003ea	:item/count	100	0xc00000003eb	FALSE
0x00001000000003ea	:item/count	250	0xc00000003eb	TRUE
0x00000c00000003ec	:db/txInstant	Thu Feb 27 19:00:00	0xc00000003ec	TRUE
0x00001000000003ea	:item/count	250	0xc00000003ec	FALSE
0x00001000000003ea	:item/count	50	0xc00000003ec	TRUE
0x00000c00000003ed	:db/txInstant	Mon Mar 31 20:00:00	0xc00000003ed	TRUE
0x00000c00000003ed	:tx/error	TRUE	0xc00000003ed	TRUE
0x00001000000003ea	:item/count	50	0xc00000003ed	FALSE
0x00001000000003ea	:item/count	9999	0xc00000003ed	TRUE
0x00000c00000003ee	:db/txInstant	Wed May 14 20:00:00	0xc00000003ee	TRUE
0x00001000000003ea	:item/count	9999	0xc00000003ee	FALSE
0x00001000000003ea	:item/count	100	0xc00000003ee	TRUE

query for transaction

built-in Db type

```
Db.V tx = Db.var();
Var<Date> txTime = var();
```

```
find(txTime)
    .where(artist.name("Paul McCartney").tx(tx),
           tx.txInstant(txTime));
```

E A V Tx

tx works like
any other entity

where are we?

a database should be	**we have seen**
powerful	unification, joins, predicates, aggregates, navigation (pull), disjunction, negation
flexible	universal schema, flat specifications, k/v, row, column, document, graph
indelible and chronological	reified transactions, time filters
in your language	an elegant Java DSL

where are we?

a database should be	**we have seen**
powerful	unification, joins, predicates, aggregates, navigation (pull), disjunction, negation
flexible	universal schema, flat specifications, k/v, row, column, document, graph
indelible and chronological	reified transactions, time filters
in your language	an elegant Java DSL **where did all the sexprs go?**

```
Query q = find(Op.count(artist).as(count))
    .where(artist.name(),
           or(artist.country(Country.US),
              artist.country(Country.CA)));
```

typed OO
DSL is icing

data (S expression)
is cake

```
[:find (count ?artist)
 :where [?artist :artist/name]
        (or [?artist :artist/country :country/US]
            [?artist :artist/country :country/US])]
```

want more dessert?

this talk was all about the icing

(brand new icing still in the kitchen)

everything you saw today (and more) available in cake-land

docs: http://docs.datomic.com/query.html

video: http://datomic.com/training.html

layering FTW

nothing in cake-land has changed

thanks!

@stuarthalloway

https://github.com/stuarthalloway/presentations/wiki
https://twitter.com/stuarthalloway
mailto:stu@cognitect.com